**Unit 14 Sequelize Homework: Reverse Engineering Code**

\*\*\* All code or file descriptions are highlighted in yellow \*\*\*

- This is middleware information for restricting routes a user is not allowed to visit if not logged in

- If the user is logged in, continue with the request to the restricted route

- If the user isn't logged in, redirect them to the login page (login.html)

**Develop > config > middleware > isAuthenticated.js**

module.exports = function(req, res, next) {

  if (req.user) {

    return next();

  }

  return res.redirect("/");

};

- This is connection configuration information that allows to connect to the server

**Develop > config > config.json**

{

  "development": {

    "username": "root",

    "password": null,

    "database": "passport\_demo",

    "host": "127.0.0.1",

    "dialect": "mysql"

  },

  "test": {

    "username": "root",

    "password": null,

    "database": "database\_test",

    "host": "127.0.0.1",

    "dialect": "mysql"

  },

  "production": {

    "username": "root",

    "password": null,

    "database": "database\_production",

    "host": "127.0.0.1",

    "dialect": "mysql"

  }

}

- File tells passport you want to login with a username/email and password

- User signs in using an email, rather than a "username"

**Develop > config > passport.js**

var passport = require("passport");

var LocalStrategy = require("passport-local").Strategy;

var db = require("../models");

passport.use(new LocalStrategy(

  {

    usernameField: "email"

  },

  function(email, password, done) {

- When the user tries to sign in this code runs

    db.User.findOne({

      where: {

        email: email

      }

    }).then(function(dbUser) {

- If there is no user with the given email this code runs

      if (!dbUser) {

        return done(null, false, {

          message: "Incorrect email."

        });

      }

- If there is a user with the given email, but the password the user gives us is incorrect

      else if (!dbUser.validPassword(password)) {

        return done(null, false, {

          message: "Incorrect password."

        });

      }

- If none of the above, return the user

      return done(null, dbUser);

    });

  }

));

- To keep authentication state across HTTP requests, Sequelize needs to serialize and dewserialize

the user

- This code is needed to make it work

passport.serializeUser(function(user, cb) {

  cb(null, user);

});

passport.deserializeUser(function(obj, cb) {

  cb(null, obj);

});

- Exports the configured passport

module.exports = passport;

- File connects to the database and retrieves login information of the user

**Develop > models > index.js**

'use strict';

var fs        = require('fs');

var path      = require('path');

var Sequelize = require('sequelize');

var basename  = path.basename(module.filename);

var env       = process.env.NODE\_ENV || 'development';

var config    = require(\_\_dirname + '/../config/config.json')[env];

var db        = {};

if (config.use\_env\_variable) {

  var sequelize = new Sequelize(process.env[config.use\_env\_variable]);

} else {

  var sequelize = new Sequelize(config.database, config.username, config.password, config);

}

fs

  .readdirSync(\_\_dirname)

  .filter(function(file) {

    return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');

  })

  .forEach(function(file) {

    var model = sequelize['import'](path.join(\_\_dirname, file));

    db[model.name] = model;

  });

Object.keys(db).forEach(function(modelName) {

  if (db[modelName].associate) {

    db[modelName].associate(db);

  }

});

db.sequelize = sequelize;

db.Sequelize = Sequelize;

- Exports the database

module.exports = db;

- File verifies user information to allow access to members page

**Develop > models > user.js**

- Requires bcrypt for password hashing.

var bcrypt = require("bcryptjs");

- Creates User model

module.exports = function(sequelize, DataTypes) {

  var User = sequelize.define("User", {

- Email can’t be null, must be an authentic email before creation

    email: {

      type: DataTypes.STRING,

      allowNull: false,

      unique: true,

      validate: {

        isEmail: true

      }

    },

    - Password can’t be null

    password: {

      type: DataTypes.STRING,

      allowNull: false

    }

  });

- Creates custom method for the User model and checks if password entered by the user can be compared to the hashed password  stored in ourdatabase

  User.prototype.validPassword = function(password) {

    return bcrypt.compareSync(password, this.password);

  };

- Hooks are automatic methods that run during various phases of the User Model lifecycle

 - Before a User is created, automatically hashes their password

  User.addHook("beforeCreate", function(user) {

    user.password = bcrypt.hashSync(user.password, bcrypt.genSaltSync(10), null);

  });

  return User;

};

- Requires user to login to proceed

**Develop > public > js > login.js**

$(document).ready(function() {

- Gets references to the form and inputs

  var loginForm = $("form.login");

  var emailInput = $("input#email-input");

  var passwordInput = $("input#password-input");

- When form is submitted, we validate there's an email and password entered

  loginForm.on("submit", function(event) {

    event.preventDefault();

    var userData = {

      email: emailInput.val().trim(),

      password: passwordInput.val().trim()

    };

    if (!userData.email || !userData.password) {

      return;

    }

- If there’s an email and password we run the loginUser function and clear the form

    loginUser(userData.email, userData.password);

    emailInput.val("");

    passwordInput.val("");

  });

- loginUser does a post to our "api/login" route and if successful, redirects us the the members page

  function loginUser(email, password) {

    $.post("/api/login", {

      email: email,

      password: password

    })

      .then(function() {

        window.location.replace("/members");

- If there's an error, log the error

      })

      .catch(function(err) {

        console.log(err);

      });

  }

});

- File gets user information and allows access to members page

**Develop > public > js > members.js**

$(document).ready(function() {

- This file just does a GET request to figure out which user is logged in and updates the HTML

on the page

  $.get("/api/user\_data").then(function(data) {

    $(".member-name").text(data.email);

  });

});

- File used to create a user for future login and access to members page

**Develop > public > js > signup.js**

$(document).ready(function() {

- Gets references to the form and input

  var signUpForm = $("form.signup");

  var emailInput = $("input#email-input");

  var passwordInput = $("input#password-input");

- When signup button is clicked,  validate the email and password aren’t blank

  signUpForm.on("submit", function(event) {

    event.preventDefault();

    var userData = {

      email: emailInput.val().trim(),

      password: passwordInput.val().trim()

    };

    if (!userData.email || !userData.password) {

      return;

    }

- If there’s an email and password, run the signUpUser function

    signUpUser(userData.email, userData.password);

    emailInput.val("");

    passwordInput.val("");

  });

- Does a post to the signup route. If successful, redirects to the members page

- Otherwise we log any errors

  function signUpUser(email, password) {

    $.post("/api/signup", {

      email: email,

      password: password

    })

      .then(function(data) {

        window.location.replace("/members");

- If there's an error, handle it by throwing up a bootstrap alert

      })

      .catch(handleLoginErr);

  }

  function handleLoginErr(err) {

    $("#alert .msg").text(err.responseJSON);

    $("#alert").fadeIn(500);

  }

});

- Adds style to the signup and login forms (login.html and signup.html)

**Develop > public > stylesheets > style.css**

form.signup,

form.login {

  margin-top: 50px;

}

- HTML layout for the login form, gets styling from style.css

**Develop > public > login.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <title>Passport Authentication</title>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">

  <link href="stylesheets/style.css" rel="stylesheet">

</head>

<body>

  <nav class="navbar navbar-default">

    <div class="container-fluid">

      <div class="navbar-header">

      </div>

    </div>

  </nav>

  <div class="container">

    <div class="row">

      <div class="col-md-6 col-md-offset-3">

        <h2>Login Form</h2>

        <form class="login">

          <div class="form-group">

            <label for="exampleInputEmail1">Email address</label>

            <input type="email" class="form-control" id="email-input" placeholder="Email">

          </div>

          <div class="form-group">

            <label for="exampleInputPassword1">Password</label>

            <input type="password" class="form-control" id="password-input" placeholder="Password">

          </div>

          <button type="submit" class="btn btn-default">Login</button>

        </form>

        <br />

        <p>Or sign up <a href="/">here</a></p>

      </div>

    </div>

  </div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

  <script type="text/javascript" src="js/login.js"></script>

</body>

</html>

- HTML layout for the members or main page

**Develop > public > members.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <title>Passport Authentication</title>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">

  <link href="stylesheets/style.css" rel="stylesheet">

</head>

<body>

  <nav class="navbar navbar-default">

  <div class="container-fluid">

    <div class="navbar-header">

      <a class="navbar-brand" href="/logout">

        Logout

      </a>

    </div>

  </div>

</nav>

  <div class="container">

    <div class="row">

      <div class="col-md-6 col-md-offset-3">

        <h2>Welcome <span class="member-name"></span></h2>

      </div>

    </div>

  </div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

  <script type="text/javascript" src="js/members.js"></script>

</body>

</html>

- HTML layout for the signup form, gets styling from style.css

**Develop > public > signup.html**

<!DOCTYPE html>

<html lang="en">

<head>

  <title>Passport Authentication</title>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">

  <link href="stylesheets/style.css" rel="stylesheet">

</head>

<body>

  <nav class="navbar navbar-default">

    <div class="container-fluid">

      <div class="navbar-header">

      </div>

    </div>

  </nav>

  <div class="container">

    <div class="row">

      <div class="col-md-6 col-md-offset-3">

        <h2>Sign Up Form</h2>

        <form class="signup">

          <div class="form-group">

            <label for="exampleInputEmail1">Email address</label>

            <input type="email" class="form-control" id="email-input" placeholder="Email">

          </div>

          <div class="form-group">

            <label for="exampleInputPassword1">Password</label>

            <input type="password" class="form-control" id="password-input" placeholder="Password">

          </div>

          <div style="display: none" id="alert" class="alert alert-danger" role="alert">

            <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>

            <span class="sr-only">Error:</span> <span class="msg"></span>

          </div>

          <button type="submit" class="btn btn-default">Sign Up</button>

        </form>

        <br />

        <p>Or log in <a href="/login">here</a></p>

      </div>

    </div>

  </div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

  <script type="text/javascript" src="js/signup.js"></script>

</body>

</html>

- This file offers a set of routes for displaying and saving data to the database

**Develop > routes > api-routes.js**

- Requires models and passport as we've configured it

var db = require("../models");

var passport = require("../config/passport");

module.exports = function(app) {

- Uses the passport.authenticate middleware with our local strategy.

- If user has valid login credentials, send them to the members page.

- Otherwise the user will be sent an error

  app.post("/api/login", passport.authenticate("local"), function(req, res) {

    res.json(req.user);

  });

- Route for signing up user. The user's password is automatically hashed and stored securely.

- If the user is created successfully, proceed to log the user in, otherwise send back an error

  app.post("/api/signup", function(req, res) {

    db.User.create({

      email: req.body.email,

      password: req.body.password

    })

      .then(function() {

        res.redirect(307, "/api/login");

      })

      .catch(function(err) {

        res.status(401).json(err);

      });

  });

- Route for logging user out

  app.get("/logout", function(req, res) {

    req.logout();

    res.redirect("/");

  });

- Route for getting some data about our user to be used client side

  app.get("/api/user\_data", function(req, res) {

    if (!req.user) {

- The user is not logged in, send back an empty object

      res.json({});

    } else {

- Otherwise send back the user's email and id

- Sending back a password, even a hashed password, isn't a good idea

      res.json({

        email: req.user.email,

        id: req.user.id

      });

    }

  });

};

- This file offers a set of routes for sending users to the various html pages

**Develop > routes > html-routes.js**

- Requiring path to so we can use relative routes to our HTML files

var path = require("path");

- Requiring our custom middleware for checking if a user is logged in

var isAuthenticated = require("../config/middleware/isAuthenticated");

module.exports = function(app) {

  app.get("/", function(req, res) {

- If the user already has an account send them to the members page

    if (req.user) {

      res.redirect("/members");

    }

    res.sendFile(path.join(\_\_dirname, "../public/signup.html"));

  });

  app.get("/login", function(req, res) {

- If the user already has an account send them to the members page

    if (req.user) {

      res.redirect("/members");

    }

    res.sendFile(path.join(\_\_dirname, "../public/login.html"));

  });

- Add our isAuthenticated middleware to this route.

- If user who is not logged in tries to access this route they will be redirected to the signup page

  app.get("/members", isAuthenticated, function(req, res) {

    res.sendFile(path.join(\_\_dirname, "../public/members.html"));

  });

};

- This file contains all information that describes the node modules tree including dependencies and

developer information 

**Develop > package.json**

{

  "name": "1-Passport-Example",

  "version": "1.0.0",

  "description": "",

  "main": "server.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start": "node server.js",

    "watch": "nodemon server.js"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "bcryptjs": "2.4.3",

    "express": "^4.17.0",

    "express-session": "^1.16.1",

    "mysql2": "^1.6.5",

    "passport": "^0.4.0",

    "passport-local": "^1.0.0",

    "sequelize": "^5.8.6"

  }

}

- This file is the initial starting point for the Node/Express server.

- File that invokes the app

**Develop > server.js**

- Requires necessary npm packages, dependencies

var express = require("express");

var session = require("express-session");

- Requires passport as it was configured

var passport = require("./config/passport");

- Sets up port and requiring models for syncing

var PORT = process.env.PORT || 8080;

var db = require("./models");

- Creates express app and configuring middleware needed for authentication

var app = express();

app.use(express.urlencoded({ extended: true }));

app.use(express.json());

app.use(express.static("public"));

- Use sessions to keep track of user's login status

app.use(session({ secret: "keyboard cat", resave: true, saveUninitialized: true }));

app.use(passport.initialize());

app.use(passport.session());

- Requires routes used

require("./routes/html-routes.js")(app);

require("./routes/api-routes.js")(app);

- Syncs database and logging a message to the user upon success

db.sequelize.sync().then(function() {

  app.listen(PORT, function() {

    console.log("==> 🌎  Listening on port %s. Visit http://localhost:%s/ in your browser.", PORT, PORT);

  });

});

**Instruction To Add / Improve Code**

**- Add a gitignore file to prevent large or unnecessary files to be pushed to Github repository.**

**- Add more css style for the login and signup pages to be more appealing.**

**- Console log all pertinent user information for easier development purposes.**

**- Change ‘var’ variable callouts to ‘const’ or ‘let’ following ES6 notation.**

**- Define password parameters to user defining a min or max length with specific character**

**allowance.**